# Warm-Starting Nested Rollout Policy Adaptation with Optimal Stopping

**Chen Dang**[1,2], **Cristina Bazgan**[2], **Tristan Cazenave**[2], **Morgan Chopin**[1], **Pierre-Henri Wuillemin**[3]

[1] Orange Labs, Châtillon, France
[2] Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, F-75016 Paris, France
[3] Sorbonne Université, CNRS, UMR 7606, LIP6, F-75005 Paris, France
chen.dang@orange.com, cristina.bazgan@dauphine.psl.eu, tristan.cazenave@dauphine.psl.eu, morgan.chopin@orange.com, pierre-henri.wuillemin@lip6.fr

## Abstract

Nested Rollout Policy Adaptation (NRPA) is an approach using online learning policies in a nested structure. It has achieved a great result in a variety of difficult combinatorial optimization problems. In this paper, we propose Meta-NRPA, which combines optimal stopping theory with NRPA for warm-starting and significantly improves the performance of NRPA. We also present several exploratory techniques for NRPA which enable it to perform better exploration. We establish this for three notoriously difficult problems ranging from telecommunication, transportation and coding theory, namely Minimum Congestion Shortest Path Routing, Traveling Salesman Problem with Time Windows and Snake-in-the-Box. We also improve the lower bounds of the Snake-in-the-Box problem for multiple dimensions.

## Introduction

Search algorithms can be used to solve many difficult combinatorial optimization problems. Monte Carlo Search algorithms rely on randomness to discover good sequences of decisions for difficult problems. Following their success in games (Coulom 2007; Kocsis and Szepesvári 2006; Browne et al. 2012; Silver et al. 2016, 2017, 2018), they were applied with success to multiple combinatorial optimization problems (Cazenave 2009; Rosin 2011). They work particularly well when combined with machine learning.

We propose new general techniques to Monte Carlo Search algorithms that improve the algorithms for multiple applications. Using these techniques we get better results than the previous ones for three difficult combinatorial optimization problems from varied fields, namely telecommunication, transportation and coding theory.

The problems we address were already successfully addressed using Nested Rollout Policy Adaptation (NRPA) (Rosin 2011). They are the Minimum Congestion Shortest Path Routing problem (Dang et al. 2021), the Traveling Salesman Problem with Time Windows (Edelkamp et al. 2013) and the Snake-in-the-Box problem (Edelkamp and Cazenave 2016). For these three problems we improve the results compared to standard NRPA. In particular for the Snake-in-the-Box problem, we provide new lower bounds for several dimensions.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Our contributions deal with the initialization of NRPA using the optimal stopping theory and with better exploration avoiding already scored sequences of decisions. More precisely, we observed that whenever NRPA gets unsatisfactory solutions from the beginning it is highly unlikely that it will find significantly better solutions in subsequent iterations. To remedy this problem, we cast the initialization step of NRPA as an instance of the secretary problem, a well-known optimal stopping problem: a decision maker (DM) wants to recruit a secretary for a job position, $n$ candidate secretaries are thus interviewed one after the other in a random order, which can be ranked among the examined candidates. The DM can decide whether to terminate the recruitment process by accepting the last interviewed candidate. The decision of DM about recruiting a candidate needs to be just after the interview of the candidate and it is irrevocable. In addition, DM has no knowledge of the quality of the upcoming candidates. The goal is to maximize the probability of selecting the best candidate. In our case, we are interested in a variant of this problem, where candidates are NRPA runs and we aim at minimizing the expected rank of the chosen candidate. To our knowledge, this is the first time that the optimal stopping theory is used in the context of Monte Carlo search.

The paper is organized in five sections. The second section deals with related works. The third section details our contributions. The fourth section gives experimental results for the Minimum Congestion Shortest Path Routing problem, the Traveling Salesman Problem with Time Windows and the Snake-in-the-Box problem. Conclusions and future works are given in section five.

## Related Works

### Monte Carlo Search

Monte Carlo Search has many applications in games and difficult combinatorial optimization problems. When combined with deep learning it surpasses the level of the best human players in games such as Go, Chess and Shogi (Silver et al. 2018). The combination has been applied to many other games with success (Cazenave et al. 2020). It is also the best general algorithm to solve a problem when given only the raw description of the problem as is the case in General Game Playing. Since 2007, all the world champions of the General Game Playing competition have used Monte Carlo

Search (Finnsson and Björnsson 2008; Méhat and Cazenave 2010).

A variant of Monte Carlo Search that works well for combinatorial optimization problems is Nested Monte Carlo Search (Cazenave 2009). It uses nested levels of playouts and memorizes the best playout of each level. It has applications in various domains, ranging from the Snake-in-the-Box (Kinny 2012) to the design of RNA molecules (Portela 2018).

## NRPA

NRPA (Rosin 2011) keeps some of the ideas from Nested Monte Carlo Search: the nested levels and the memorization of the best sequence at each level. However, it differs from Nested Monte Carlo Search as it learns a playout policy specific to the instance it solves and as it does not recursively search the state space. It only explores the state space with non-uniform playouts following the learned policy.

NRPA has been applied to numerous problems. Its first results were worlds records in Morpion Solitaire and Crosswords Puzzles (Rosin 2011). It was then applied to transportation problems such as the Traveling Salesman Problem with Time Windows (Cazenave and Teytaud 2012a; Edelkamp et al. 2013) and the Vehicle Routing Problems (Abdo, Edelkamp, and Lawo 2016; Edelkamp and Cazenave 2016; Cazenave et al. 2021; Sentuc, Cazenave, and Lucas 2022). It was also applied to network problems such as Network Traffic Engineering (Dang et al. 2021) or Virtual Network Embedding (Elkael et al. 2021). Other applications include applications in bioinformatics such as Multiple Sequence Alignment (Edelkamp and Tang 2015) or the design of RNA molecules (Cazenave and Fournier 2020).

Multiple improvements have been proposed to NRPA. An improvement that works well for problems such as Weak Schur Numbers, SameGame, and the Bus Regulation problem is Selective NRPA (Cazenave 2016). It avoids playing actions that are considered bad in the playouts and results in a possibly large increase in the scores of the playouts. A generalization of this behavior is Generalized NRPA (Cazenave 2020) that uses a bias on the possible actions to perform playouts with an informed policy. It is a generalization of NRPA since GNRPA with a bias of zero for all actions is NRPA and it is a generalization of Selective NRPA since a bias of $-\infty$ for actions to avoid is equivalent to Selective NRPA. GNRPA improves a lot on NRPA for problems such as the Traveling Salesman with Time Windows and Vehicle Routing Problems. Yet another improvement to NRPA is Beam NRPA (Cazenave and Teytaud 2012b) that memorizes multiple best sequences per level instead of one. Combined with a measure of the diversity of the sequences it has led to Diversity NRPA (Edelkamp and Cazenave 2016) that broke records in the Snake-in-the-Box and the Vehicle Routing Problems.

The NRPA algorithm is shown in Algorithms 1, 2 and 3.

**Algorithm 1:** The playout algorithm

```
Function playout (state, weight):
    $sequence \leftarrow []$
    while state is not terminal do
        $z \leftarrow \sum_{a' \in \mathcal{A}(state)} e^{weight(a')}$
        Draw $a$ with probability $\frac{1}{z} e^{weight(a)}$
        $state \leftarrow$ play$(state, a)$
        append $a$ to $sequence$
    end
    return (score$(state)$, $sequence$)
```

**Algorithm 2:** The adapt algorithm

```
Function adapt (weight, sequence, α):
    $w \leftarrow weight$
    $state \leftarrow root$
    for $a$ in sequence do
        $z \leftarrow \sum_{a' \in \mathcal{A}(state)} e^{weight(a')}$
        for $a' \in \mathcal{A}(state)$, do
            $w(a') -= \alpha \cdot \frac{1}{z} e^{weight(a')}$
        end
        $w(a) += \alpha$
        $state \leftarrow$ play$(state, a)$
    end
    return $w$
```

**Algorithm 3:** The NRPA algorithm

```
Function NRPA (level, weight):
    if $level == 0$ then
        return playout$(root, weight)$
    else
        $bestScore \leftarrow \infty$
        for N iterations do
            $(sc, new) \leftarrow$ NRPA$(level - 1, weight)$
            if $sc \leq bestScore$ then
                $bestScore \leftarrow sc$
                $seq \leftarrow new$
            end
            $weight \leftarrow$ adapt$(weight, seq, \alpha)$
        end
    end
    return $(bestScore, seq)$
```

## The Secretary Problem

First introduced in the early '60s, the secretary problem is a famous optimal stopping problem: a decision maker (DM) wants to recruit a secretary for a job position, $n$ candidate secretaries are thus interviewed one after the other in a random order, which can be ranked among the examined candidates. The DM can decide whether to terminate the recruitment process by accepting the last interviewed candidate. The decision of DM about recruiting a candidate needs to be just after the interview of the candidate and it is irrevocable. In addition, DM has no knowledge of the quality of the upcoming candidates.

The classical secretary problem attempts to maximize the probability of selecting the best candidate. In our case, we are interested in a variant of this problem, which aims to minimize the expected rank of the chosen candidate. (Chow et al. 1964) showed that the optimal stopping rule satisfies $\prod_{j=1}^{n}(\frac{j+2}{j})^{1/j+1} \cong 3.8695$, which can be determined by dynamic programming. In practice, this is hard to implement for large $n$. (Krieger and Samuel-Cahn 2009) proposed much simpler stopping rules that perform almost equally well in minimizing the sum of the expected rankings of the chosen items when one or more candidates are desired.

When only one candidate is needed, with stopping rule $R_i \leq \frac{ic}{n+1-i}$ for a given constant $c$, where $R_i$ is the relative rank of the $i$th item and $n$ the total number of items, we can have an asymptotic expected absolute rank of 3.928 for the chosen candidate. When multiple candidates are to be retained, we are interested in the case where a given percentage of candidates are desired. For a fixed given percentage $\alpha$, $0 < \alpha < 1$, the item $j$ is kept if it satisfies the condition $R_i \leq \lceil \alpha i \rceil$. With this simple rule, we have $\frac{S_n}{nL_n} \to \frac{\alpha}{2}$ a.s. as $n \to \infty$, where $L_n$ is the number of candidates retained, $S_n$ the sum of the absolute ranks of the candidates retained.

## Contributions

### Meta-NRPA

Inspired by the secretary problem, we hereby present Meta-NRPA, which warm-starts the NRPA with optimal stopping theory.

When dealing with difficult combinatorial optimization problems with large instances, sometimes the variance between executions of NRPA can be very important. After being stuck in the local minimum, it's difficult to escape to a better solution. If NRPA is optimized towards a better local minimum from the beginning, we can easily get better results at the end of the execution. On the contrary, if it gets unsatisfactory results from the beginning, there is only a small probability that it will get better results in subsequent execution than in other executions (see for instance Figure 3).

From this observation, we have the idea of warm-starting the executions of NRPA by selecting those that lead to better results and discarding those that do not. The choice is made according to the performance of a small fraction of the NRPA runs at the beginning. If it satisfies the criteria of the optimal stopping, the NRPA is continued to its full execution, otherwise, the current execution is abandoned and a new NRPA is executed.

When there is not enough time, only several complete level $l$ NRPA executions can be performed, and the optimal stopping rule with only one chosen candidate is used. In practice, we execute NRPA with level $l-1$ and compare its score with existing scores. If the relative rank $R_i$ of the $i$-th NRPA satisfies $R_i \leq \frac{ic}{n+1-i}$, we continue the current NRPA with level $l$; otherwise, it's abandoned and next NRPA with level $l-1$ is performed and compared, until the timeout. The algorithm is shown in Algorithm 4. The function *ContinueNrpa* executes a normal NRPA, but with the best score and the best sequence of previous NRPA run of level $l-1$.

When many executions of the NRPA are possible within the time limit, we tend to select a certain percentage of executions to continue, as this will allow the NRPA to be optimized more deeply in a variety of different directions. The algorithm is shown in Algorithm 5. Figure 1 demonstrates the process of this version of Meta-NRPA with level 2.

**Algorithm 4:** Meta-NRPA with one item

```
Function MetaNrpa(level):
    $scores$ ← empty ordered list
    $i \leftarrow 0$
    while not timeout do
        $weights$ ← InitPolicy()
        $score, seq$ ← NRPA($level - 1$, $weights$)
        $R_i$ ← $scores$.insert($score$)
        $i+ = 1$
        $n$ ← NbExecutionEstimation()
        if $R_i \leq \frac{ic}{n+1-i}$ then
            $score$ ← ContinueNrpa($level$, $weights$)
            $scores$.insert($score$)
            $i+ = 1$
        end
    end
    return min($scores$)
```

**Algorithm 5:** Meta-NRPA with $\alpha\%$ items

```
Function MetaNrpa(level):
    $scores$ ← empty ordered list
    $i \leftarrow 0$
    while not timeout do
        $weights$ ← InitPolicy()
        $score, seq$ ← NRPA($level - 1$, $weights$)
        $R_i$ ← $scores$.insert($score$)
        $i+ = 1$
        if $R_i \leq \lceil \alpha i \rceil$ then
            $score, seq$ ← ContinueNrpa($level$,
              $weights$)
            $scores$.insert($score$)
            $i+ = 1$
        end
    end
    return min($scores$)
```

Note that the assumption we apply here is that NRPA that performs well at the beginning will also provide better results in the later optimizations. This is not always true, but statistically speaking, this assumption holds most of the time. And the larger the fraction of NRPA we run, the more likely this assumption will hold.

Using a larger fraction of the NRPA strengthens this assumption, but it also takes more time, so we get fewer candidates in a fixed period and therefore risk losing the opportunity to get better candidates. In fact, running only the inferior level of NRPA in our tests has roughly shown a strong correlation with the final NRPA results and has been able to obtain satisfactory results.

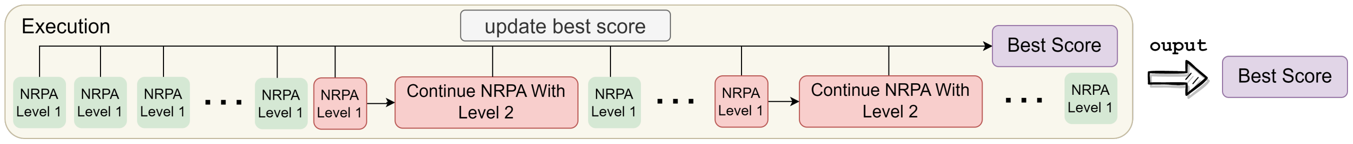


Figure 1: Meta-NRPA. We execute NRPA with level $l-1$ and compare its score with existing scores. If the relative rank $R_i$ of the $i$-th NRPA satisfies a certain condition, we continue the current NRPA with level $l$; otherwise, it's abandoned and the next NRPA with level $l-1$ is performed and compared, until the timeout. The final score of Meta-NRPA is the best score among all executions of NRPA.

## Exploratory NRPA

Since NRPA tends to converge quickly and has difficulty getting away from local minima, we also use several techniques to help NRPA do more exploration. These techniques do not work for every problem, but in our tests, they help a lot for one or many problems.

**Force Exploration** Force exploration was introduced by (Dang et al. 2021) and greatly improves the performance of NRPA by performing random selection when an already discovered solution is encountered. This simple mechanism encourages exploration while maintaining the original structure and strategy learning of NRPA.

**$\epsilon$-greedy NRPA** When the NRPA is executed after some time, the NRPA may converge to some specific solution and keep exploiting the same solution, which does not help to escape the local minima. Inspired by the $\epsilon$-greedy agent in multi-armed bandit problem (Sutton and Barto 2018), we propose here $\epsilon$-greedy NRPA. $\epsilon$-greedy NRPA uses the same idea as a $\epsilon$-greedy agent: each choice is made with uniform probability for a probability of $\epsilon$, while following the learned policy of NRPA with a probability of $1-\epsilon$. It allows NRPA to keep exploring at any stage of the execution, giving better performance for long-running executions.

**Dynamic Learning Rate** Without finding a better solution, NRPA continues to enforce the policy on the best-known sequence at the current level, which can easily lead to a rapid convergence to a sub-optimal value. As a result, the exploration of other options decreases, which makes it more difficult to find better results.

To address this problem, we propose a dynamic learning rate that reduces the learning rate of the NRPA if it has difficulty finding a better solution. In this way, NRPA still converges quickly towards the best-known solution at the beginning but slows down when another better solution is not found, thus encouraging exploration. The modified learning rate $\alpha_{new}$ is defined as:

$$\alpha_{new} = \frac{\alpha}{N_{stagnant}}$$

where $\alpha$ is the default learning rate, $N_{stagnant}$ is the number of times that the playout result is not better than the best result at the current level.

## Experimental Results

In this section, we test the efficiency of our approach on the following three notoriously NP-hard optimization problems.

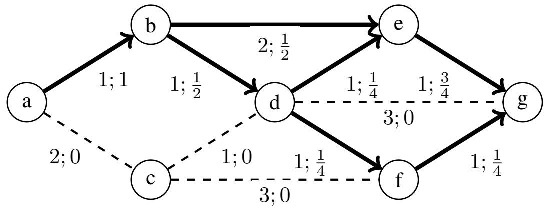


Figure 2: Illustration of a shortest path routing with the ECMP rule. In this figure, we assume unit capacities and suppose that a demand $k$ with traffic volume $D^k = 1$ must be routed from $s^k = a$ to $t^k = g$. A label $w_{uv}; load(uv, w)$ is associated to each arc $uv \in A$.

## Minimum Congestion Shortest Path Routing

Among the optimization problems arising in telecommunication traffic engineering (TE), we address the one related to setting weights in networks that are based on shortest path routing protocols (e.g OSPF, IS-IS). Indeed, finding weights that induce efficient routing paths (e.g that minimize the maximum congested link) is a well-known and well-studied problem in TE (the reader is referred to (Bley et al. 2010) for more details on the subject).

Formally, we consider the Minimum Congestion Shortest Path Routing problem that takes as input a bidirected graph $G = (V, A)$ whose vertices correspond to routers and the arcs correspond to links between routers. Every arc $uv$ is associated with a capacity denoted by $c_{uv}$. Let $K$ denote a set of demands or commodities to be routed in $G$. Each demand $k \in K$ is defined by a pair of vertices $s^k$ and $t^k$ representing the source and the target of $k$, a traffic volume $D^k$ to be routed from $s^k$ to $t^k$. Such a demand $k$ will be denoted by the quadruplet $(s^k, t^k, D^k)$. Given a metric $w \in Z_+^{|A|}$, each demand $k \in K$ is routed along the shortest paths between $s^k$ and $t^k$. If there is more than one shortest path joining the extremities of $k$, the traffic volume $D^k$ is split evenly among those paths according to the so-called ECMP (Equal-Cost Multi-Path) rule. More precisely, the traffic volume that reaches a node $v \in V$ must be split equally among all arcs leaving $v$ and belonging to the shortest paths toward destination $t^k$. We then define the *load* of an arc $uv$ induced by $w$, denoted by $load(uv, w)$, as the amount of traffic traversing the arc $uv$ over its capacity (see Figure 2). The *congestion* $cong(w)$ of a given metric $w$ is defined by $\max_{uv \in A} load(uv, w)$, that is the maximum load over all arcs. The problem asks to find a metric $w \in Z_+^{|A|}$ and the

| name | $\|V\|$ | $\|A\|$ | $\|K\|$ | $\sum D^k$ |
|---|---|---|---|---|
| igen50 | 50 | 101 | 1221 | 34517 |
| igen100 | 100 | 200 | 2399 | 83248 |
| igen500 | 500 | 1009 | 12565 | 377072 |
| igen1000 | 1000 | 2025 | 14908 | 487523 |
| igen2000 | 2000 | 4028 | 14918 | 488331 |
| igen5000 | 5000 | 10029 | 19863 | 656416 |
| inet3040 | 3040 | 9586 | 4540 | 144103 |
| inet4000 | 4000 | 13288 | 7920 | 266207 |
| inet5000 | 5000 | 17380 | 12717 | 414545 |

Table 1: Information of graphs: network name, number of nodes, number of arcs, number of demands, total volume of demands

routing paths induced by these weights such that the network congestion $cong(w)$ is minimum.

Table 1 shows all the graphs we used for the experiments. Two popular large-scale telecommunication topology generators are used, IGen (Quoitin et al. 2009) and Inet (Winick and Jamin 2002). IGen generates two-level networks with a backbone node and access node topology, while Inet follows a power-law distribution of node degrees, which corresponds to the topology of the Internet.

For the traffic matrix, we generate traffic between any two nodes with a fixed probability, and the volume of the demand $D^k$ for demand $k$ between nodes $s$ and $t$ is :

$$D^k = \alpha S_s T_t C_{(s,t)} e^{\frac{-d(s,t)}{2d_{\max}}}$$

where $\alpha$ is a constant, $S_u, T_u \in [0, 1]$ are two random numbers for node $u$, $C_{(s,t)} \in [0, 1]$ is a random number for node pair $(s, t)$, $d(s, t)$ is the distance between node $s$ and $t$, $d_{max}$ is the maximum distance between any pair of nodes in the graph.

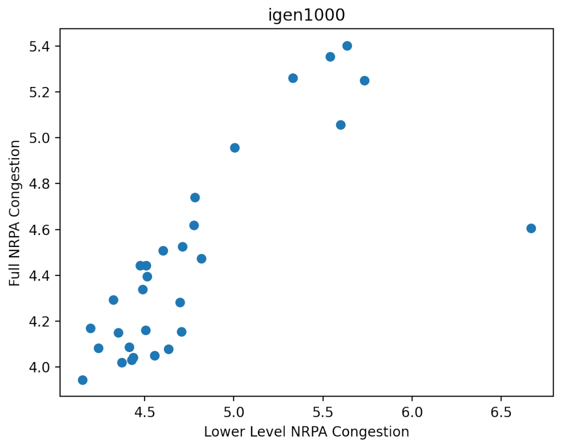
Finally, we define the score used in NRPA to evaluate a solution $w \in Z_+^{|A|}$ as the congestion value induced by $w$, that is:

$$score(w) = cong(w)$$

Figure 3 shows an example of the correlation of the congestion values between the 30 independent runs of small fraction of the NRPA executed in Meta-NRPA and the congestion value of the full NRPA execution. This confirms the earlier assumption that, in general, NRPAs that perform better at the beginning provide better results when they are fully executed.

Figure 4 shows the comparison between the congestion values obtained with NRPA and Meta-NRPA with one item. Our implementation of NRPA is similar as in (Dang et al. 2021) except that we use the Customizable Contraction Hierarchy (CCH) algorithm (Dibbelt, Strasser, and Wagner 2016), instead of Dijkstra, for improving the computation time of the shortest paths and, hence, the congestion value.

Since each playout takes a long time, we use NRPA with a level of 2 and 50 iterations. Each method executes 20 independent runs on each graph, the results are normalized according to the lower bound calculated by Fleischer's approximation scheme with $\epsilon = 0.1$ (Fleischer 2000). Graphs



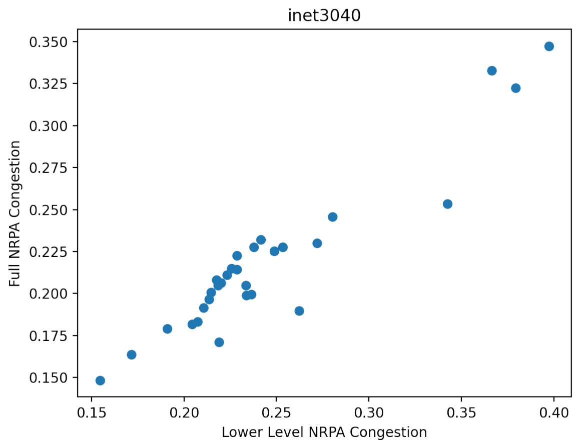


Figure 3: Correlation between the congestion values of the lower level NRPA and the full NRPA of 30 independent executions on two generated telecommunication networks igen1000 and inet3040. The full NRPA execution is the continuation of the lower level NRPA.

having more than 400 nodes are executed for 2 hours, others for 30 minutes. We do not use $\epsilon$-greedy or dynamic learning rate, because for this problem NRPA already has difficulty converging within the time limit.

The results clearly show that Meta-NRPA can help reduce the congestion values on all instances and can lead to a reduction in the variance between executions.

## Traveling Salesman Problem with Time Windows

In the Traveling Salesman Problem (TSP) a set of $n$ cities (one of which is the depot) and their pairwise distances are given. The task is to find the shortest route that starts and ends at the depot and visits each city only once. In the Traveling Salesman Problem with Time Windows (TSPTW), in addition to the conditions of the TSP, each city must be visited and left within a given time interval, which is much more difficult than the TSP.

The TSPTW can be formulated as follow. An undirected

| T (min) | NRPA | NRPAv1 | NRPAv2 | NRPAv3 | NRPAv4 |
|---|---|---|---|---|---|
| 1 | -30914.67 | -10913.89 | -910.06 | -893.14 | **-891.30** |
| 5 | -30914.45 | -7580.00 | -901.19 | -893.11 | **-888.92** |
| 10 | -30914.45 | -7579.77 | -897.30 | -893.11 | **-887.54** |
| 30 | -30914.12 | -7579.48 | -892.33 | -893.09 | **-886.80** |

Table 2: Average score of 30 independent executions of Exploratory NRPA with TSPTW on rc204.1 instance. NRPAv1: NRPA + force exploration, NRPAv2: NRPA + force exploration + greedy, NRPAv3: NRPA + force exploration + dynamic alpha, NRPAv4: NRPA + force exploration + dynamic alpha + greedy

| T (min) | Meta-NRPA | Meta-NRPAv1 | Meta-NRPAv2 | Meta-NRPAv3 | Meta-NRPAv4 |
|---|---|---|---|---|---|
| 1 | -4241.87 | -908.41 | -903.19 | -892.37 | **-888.03** |
| 5 | -901.01 | -903.69 | -895.45 | -887.53 | **-884.73** |
| 10 | -896.53 | -898.84 | -892.93 | -884.71 | **-883.00** |
| 30 | -890.84 | -890.49 | -887.15 | -881.68 | **-881.22** |

Table 3: Average score of 30 independent executions of Meta-NRPA with TSPTW on rc204.1 instance

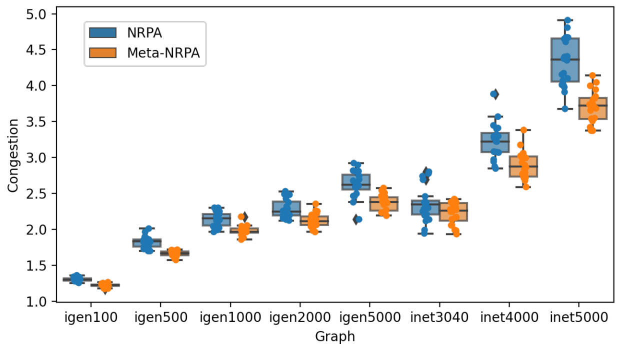


Figure 4: Congestion value comparison of 20 independent executions of NRPA and Meta-NRPA on large generated telecommunication network instances

graph $G(N, A)$ represents the cities and the corresponding transportation network, where the nodes $N = n_1, n_2, ...n_{|N|}$ corresponds to the cities, among which the node $n_1$ corresponds to the depot, and $A = N \times N$ represents to the edges between nodes. The distance between two cities $n_i$ and $n_j$ is also the cost function $c(n_i, n_j)$. We aim to minimize the total cost function:

$$cost(P) = \sum_{k=1}^{|N|} c(p_k, p_{k+1})$$

where $P = (p_1, \ldots, p_{|N|+1})$, $(p_1, \ldots, p_{|N|})$ is a permutation of $\{n_1, n_2, \ldots, n_{|N|}\}$ and $p_{|N|+1} = p_1 = n_1$.

Apart from the distance cost, a time window is also added as a constraint. For every city $n_i$, we have a time interval $[e_i, \ell_i]$, which means it must be visited before $\ell_i$, but if it's visited before $e_i$, we must wait until $e_i$ to depart. We can set every violation of the constraint as a penalty of one million in the score. So the final score of path $P$ is:

$$score(P) = -cost(P) - 10^6 \times \Omega(P)$$

where $\Omega(P)$ is the number of violated constraints.

NRPA-based algorithms have been used on this problem and have achieved good results (Cazenave 2020).

We test our algorithms on rc204.1, which is the most difficult instance in the Solomon-Potwin-Bengio TSPTW benchmark. We use NRPA of level 4 and 100 iterations. The inverse distance between every two cities is used to initialize the NRPA policy. The learning rate of NRPA $\alpha$ is set to 0.01 to avoid premature convergence.

Table 2 shows the comparison of different versions of exploratory NRPA. Each result is the average score of 30 independent executions. We can see the original NRPA struggles to make sure every solution satisfies the constraints, while force exploration greatly improved the performance of NRPA. With $\epsilon$-greedy and dynamic learning rate, NRPA achieves much better performance.

Table 3 shows the corresponding Meta-NRPA of every version of exploratory NRPA shown in Table 2. Because NRPA takes a relatively long time on each execution, we use Meta-NRPA with 30% items. We can see that for every version, Meta-NRPA significantly improves the score. And Meta-NRPA with all the exploratory techniques achieves the best result among all variants.

## Snake-in-the-Box

The Snake-in-the-Box problem is that of finding the longest induced path in an $n$-dimensional hypercube, which is a special case of the induced subgraph isomorphism problem. The path starts at one node of the hypercube and moves as far as possible along the edge to a neighbor node. Each time it moves one step to a new node, the previous node, and its neighboring nodes become unavailable and can never be traveled to. The detailed description of the problem can be found in (Abbott and Katchalski 1988). (Potter et al. 1994), (Östergård and Pettersson 2014), (Wynn 2012) and (Kinny 2012) respectively gave the records on this problem on dimensions 7, 8, 9 and 10, and (Allison and Paulusma 2016) has the latest record on dimensions 11, 12, and 13.

In this problem, we define the score used in NRPA to eval-

| d | NRPA | NRPAv1 | NRPAv2 | NRPAv3 | NRPAv4 | Meta-NRPAv2 |
|---|---|---|---|---|---|---|
| 8 | 95.55 | **97.00** | **97.00** | **97.00** | **97.00** | **97.00** |
| 9 | 175.55 | 185.30 | 186.40 | 182.40 | 184.90 | **186.55** |
| 10 | 333.60 | 350.50 | 357.25 | 352.50 | 355.90 | **358.65** |
| 11 | 612.75 | 628.05 | 668.05 | 647.75 | 597.00 | **668.35** |

Table 4: Average score of 20 independent executions of 10 minutes for the Snake-in-the-Box problem. NRPAv1: NRPA + force exploration, NRPAv2: NRPA + force exploration + greedy, NRPAv3: NRPA + force exploration + dynamic alpha, NRPAv4: NRPA + force exploration + dynamic alpha + greedy

| Dimension | Meta-NRPA-fe | Best Known Score |
|---|---|---|
| 7 | 50 | 50 |
| 8 | 97 | 98 |
| 9 | 188 | 190 |
| 10 | **373** | 370 |
| 11 | **721** | 712 |
| 12 | **1383** | 1373 |
| 13 | **2709** | 2687 |

Table 5: Comparison of Meta-NRPA with known lower bounds on the Snake-in-the-Box

uate a path $P$ as the number of edges of $P$:

$$score(P) = |P|$$

Since, by definition, whenever a node is visited all its neighbors are unavailable, it is preferable to visit nodes with fewer available neighboring nodes to increase the chance of finding a long path. A prior is thus added to prefer the nodes with fewer available neighbors.

Table 4 compares the performance of NRPAs. Each result is the average score of 20 independent runs of 10 minutes. We also used the best sequence of the lower dimension as primers for the NRPAs. Each playout will first apply the primer, and try to extend the sequence. With force exploration, NRPA has the possibility of changing the primer after finding that the current sequence has already been discovered. We use NRPA with level 4, 100 iterations, Meta-NRPA with 10% items, and 5% for $\epsilon$-greedy, 0.01 for learning rate $\alpha$.

For this problem, NRPA with $\epsilon$-greedy and dynamic learning rate have a similar effect of improving the score, but $\epsilon$-greedy performs significantly better than dynamic learning rate. Using all two techniques leads to worse performance than using only $\epsilon$-greedy as it would cause over-exploration. From the result, we can see that NRPA with force exploration, $\epsilon$-greedy, and Meta-NRPA has the best performance among all variants.

We also used another slightly different approach to try to achieve a new record for this problem. At each execution, the current record in each dimension is used as a primer. We only used Meta-NRPA with force exploration and did not use $\epsilon$-greedy or dynamic learning rate.

As a result of the force exploration, Meta-NRPA will apply the best sequence before changing one random choice in the sequence to another random valid value and following the policy for the rest of the choices. We run 20 executions in parallel over 30 minutes. Each time we have a new record, it is used as a new primer, and a new execution is performed. We kept the experiment going until we cannot get better results.

Table 5 shows the comparison of our record of dimension 10, 11, 12 and 13 and the best known record (Allison and Paulusma 2016). For dimensions 7, 8 and 9 we don't have a new record, so we show our result using a inferior dimension record as a primer.

## Conclusion

In this paper, we introduced Meta-NRPA algorithm along with several exploratory techniques that substantially improve the performance of NRPA. More precisely, using optimal stopping theory and avoiding already seen sequences of decision we got better results for three difficult problems from varied fields.

We improved the lower bounds for the Snake-in-the-Box problem with our Meta-NRPA.

The new techniques we proposed are general improvements to NRPA and could be applied to many other difficult combinatorial optimization problems.

## References

Abbott, H.; and Katchalski, M. 1988. On the snake in the box problem. *Journal of Combinatorial Theory, Series B*, 45(1): 13–24.

Abdo, A.; Edelkamp, S.; and Lawo, M. 2016. Nested rollout policy adaptation for optimizing vehicle selection in complex VRPs. In *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*, 213–221. IEEE.

Allison, D.; and Paulusma, D. 2016. New Bounds for the Snake-in-the-Box Problem. *ArXiv*, abs/1603.05119.

Bley, A.; Fortz, B.; Gourdin, É.; Holmberg, K.; Klopfenstein, O.; Pióro, M.; Tomaszewski, A.; and Ümit, H. 2010. Optimization of OSPF Routing in IP Networks. In *Graphs and Algorithms in Communication Networks*, 199–240. Springer.

Browne, C.; Powley, E.; Whitehouse, D.; Lucas, S.; Cowling, P.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1): 1–43.

Cazenave, T. 2009. Nested Monte-Carlo Search. In Boutilier, C., ed., *IJCAI*, 456–461.

Cazenave, T. 2016. Nested rollout policy adaptation with selective policies. In *Computer Games*, 44–56. Springer.

Cazenave, T. 2020. Generalized nested rollout policy adaptation. In *Monte Carlo Search International Workshop*, 71–83. Springer.

Cazenave, T.; Chen, Y.-C.; Chen, G.-W.; Chen, S.-Y.; Chiu, X.-D.; Dehos, J.; Elsa, M.; Gong, Q.; Hu, H.; Khalidov, V.; Cheng-Ling, L.; Lin, H.-I.; Lin, Y.-J.; Martinet, X.; Mella, V.; Rapin, J.; Roziere, B.; Synnaeve, G.; Teytaud, F.; Teytaud, O.; Ye, S.-C.; Ye, Y.-J.; Yen, S.-J.; and Zagoruyko, S. 2020. Polygames: Improved Zero Learning. *ICGA Journal*, 42(4): 244–256.

Cazenave, T.; and Fournier, T. 2020. Monte Carlo inverse folding. In *Monte Carlo Search International Workshop*, 84–99. Springer.

Cazenave, T.; Lucas, J.-Y.; Triboulet, T.; and Kim, H. 2021. Policy adaptation for vehicle routing. *Ai Communications*, 34(1): 21–35.

Cazenave, T.; and Teytaud, F. 2012a. Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In *International Conference on Learning and Intelligent Optimization*, 42–54. Springer.

Cazenave, T.; and Teytaud, F. 2012b. Beam nested rollout policy adaptation. In *Computer Games Workshop at ECAI*.

Chow, Y. S.; Moriguti, S.; Robbins, H.; and Samuels, S. M. 1964. Optimal Selection Based on Relative Rank (the “Secretary Problem”). *Israel Journal of Mathematics*, 2(2): 81–90.

Coulom, R. 2007. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, 72–83. Springer.

Dang, C.; Bazgan, C.; Cazenave, T.; Chopin, M.; and Wuillemin, P.-H. 2021. Monte Carlo Search Algorithms for Network Traffic Engineering. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021*, 486–501. Springer-Verlag.

Dibbelt, J.; Strasser, B.; and Wagner, D. 2016. Customizable Contraction Hierarchies. *ACM J. Exp. Algorithmics*, 21.

Edelkamp, S.; and Cazenave, T. 2016. Improved diversity in nested rollout policy adaptation. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, 43–55. Springer.

Edelkamp, S.; Gath, M.; Cazenave, T.; and Teytaud, F. 2013. Algorithm and knowledge engineering for the TSPTW problem. In *2013 IEEE Symposium on Computational Intelligence in Scheduling (CISched)*, 44–51. IEEE.

Edelkamp, S.; and Tang, Z. 2015. Monte-Carlo Tree Search for the Multiple Sequence Alignment Problem. In Lelis, L.; and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel*, 9–17. AAAI Press.

Elkael, M.; Castel-Taleb, H.; Jouaber, B.; Araldo, A.; and Aba, M. A. 2021. Improved monte carlo tree search for virtual network embedding. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 605–612. IEEE.

Finnsson, H.; and Björnsson, Y. 2008. Simulation-Based Approach to General Game Playing. In *AAAI*, 259–264.

Fleischer, L. 2000. Approximating Fractional Multicommodity Flow Independent of the Number of Commodities. *SIAM J. Discret. Math.*, 13(4): 505–520.

Kinny, D. 2012. A New Approach to the Snake-In-The-Box Problem. In *Proceedings of the 20th European Conference on Artificial Intelligence*, ECAI'12, 462–467. NLD: IOS Press. ISBN 9781614990970.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *17th European Conference on Machine Learning (ECML'06)*, volume 4212 of *LNCS*, 282–293. Springer.

Krieger, A.; and Samuel-Cahn, E. 2009. The secretary problem of minimizing the expected rank: a simple suboptimal approach with generalizations. *Advances in Applied Probability*, 41: 1041–1058.

Méhat, J.; and Cazenave, T. 2010. Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4): 271–277.

Portela, F. 2018. An unexpectedly effective Monte Carlo technique for the RNA inverse folding problem. *BioRxiv*, 345587.

Potter, W. D.; Robinson, R. W.; Miller, J. A.; Kochut, K. J.; and Redys, D. Z. 1994. Using the Genetic Algorithm to Find Snake-in-the-Box Codes. In *Proceedings of the 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, IEA/AIE '94, 421–426. USA: Gordon and Breach Science Publishers, Inc. ISBN 2884491287.

Quoitin, B.; Van den Schrieck, V.; Francois, P.; and Bonaventure, O. 2009. IGen: Generation of router-level Internet topologies through network design heuristics. In *2009 21st International Teletraffic Congress*, 1–8.

Rosin, C. D. 2011. Nested Rollout Policy Adaptation for Monte Carlo Tree Search. In *IJCAI*, 649–654.

Sentuc, J.; Cazenave, T.; and Lucas, J.-Y. 2022. Generalized Nested Rollout Policy Adaptation with Dynamic Bias for Vehicle Routing. In *AI for Transportation at AAAI*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature*, 550(7676): 354–359.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Winick, J.; and Jamin, S. 2002. Inet-3.0: Internet Topology Generator. Technical Report UM-CSE-TR-456-02, EECS, University of Michigan.

Wynn, E. 2012. Constructing circuit codes by permuting initial sequences. *arXiv*, abs/1201.1647.

Östergård, P.; and Pettersson, V. 2014. Exhaustive Search for Snake-in-the-Box Codes. *Graphs and Combinatorics*, 31.